Simulating Elementary Active Membranes
With an Application to the P Conjecture*

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,
Antonio E. Porreca, and Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati, luca.manzoni, mauri, porreca, zandron}@disco.unimib.it

Abstract. The decision problems solved in polynomial time by P systems with elementary active membranes are known to include the class $\mathbf{P}^{\#\mathbf{P}}$. This consists of all the problems solved by polynomial-time deterministic Turing machines with polynomial-time counting oracles. In this paper we prove the reverse inclusion by simulating P systems with this kind of machines: this proves that the two complexity classes coincide, finally solving an open problem by Păun on the power of elementary division. The equivalence holds for both uniform and semi-uniform families of P systems, with or without membrane dissolution rules. Furthermore, the inclusion in $\mathbf{P}^{\#\mathbf{P}}$ also holds for the P systems involved in the P conjecture (with elementary division and dissolution but no charges), which improves the previously known upper bound **PSPACE**.

1 Introduction

The computational power of P systems with elementary active membranes working in polynomial time was first investigated in [10]. The ability of P systems to exploit parallelism to perform multiple computations at the same time allows an efficient solution of **NP**-complete problems. This feature was further exploited [8] to show that P systems with elementary active membranes can solve all $\mathbf{P}^{\mathbf{PP}}$ problems (or $\mathbf{P}^{\#\mathbf{P}}$ problems, due to the equivalence of the two classes).

While all the previous results showed an ever increasing lower bound for the power of this class of P systems, the upper bound for their computational ability was proved to be **PSPACE** [9]. Therefore, until now it was only known that this class is located between $\mathbf{P}^{\#\mathbf{P}}$ and **PSPACE**. In this paper we show that the already known lower bound is, in fact, also an upper bound. This implies that non-elementary membrane division is necessary in order to solve **PSPACE**-complete problems (unless $\mathbf{P}^{\#\mathbf{P}} = \mathbf{PSPACE}$), as conjectured by Sosík and Pérez-Jiménez and formulated as Problem B by Păun in 2005 [5]. This bound

* This work was partially supported by Università degli Studi di Milano-Bicocca, FA 2013: "Complessità computazionale in modelli di calcolo bioispirati: Sistemi a membrane e sistemi di reazioni".

has also an interesting implication for the *P conjecture* [5, Problem F], stating that P systems with elementary division and dissolution but no charges can solve only problems in **P**. The previously known upper bound for the computational power of that class of P systems was also **PSPACE** but, since those systems are a weaker version of the ones studied here, the $\mathbf{P}^{\#\mathbf{P}}$ bound also applies to them.

The main idea behind the simulation of P systems with elementary active membranes by Turing machines with **#P** oracles is similar to one from [9]: we cannot store the entire configuration of the P system, since it can grow exponentially in time due to elementary membrane division. Therefore, instead of simulating directly the behaviour of the elementary membranes, we only simulate the interactions between them and their parent regions. Indeed, from the point of view of a non-elementary membrane, all the membranes it contains are just "black boxes" that absorb and release objects. We thus only store the configurations of non-elementary membranes and, when needed, we exploit a **#P** oracle to determine how many instances of each type of object are exchanged between elementary membranes and their parent regions. As will be clear in the following, while doing so we also need to take special care for send-in rules, since they require, in some sense, a partial knowledge of the parent's multiset of objects, and conflicts between send-in rules competing for the same objects that can be applied to different membranes may be difficult to resolve. Thus, we have devised a way to provide a "centralised control" that allows to correctly apply the different send-in rules.

The paper is structured as follows. In Section 2 the basic definitions concerning P systems and the relevant complexity classes are briefly recalled. The simulation of P systems is described in Section 3; in particular, the three phases of the simulation algorithm requiring more computing power than a deterministic Turing machine working in polynomial time are detailed in Section 3.1 (movement of objects into elementary membranes), Section 3.2 (releasing objects from elementary membranes), and Section 3.3 (establishing if a membrane is elementary). The theorem stating the main result and the implications for the P conjecture are presented in Section 4. The paper is concluded with a summary of the results and some directions for future research in Section 5.

2 Basic Notions

We begin by recalling the basic definition of P systems with (elementary) active membranes [4].

Definition 1. *A* P system with elementary active membranes *of initial degree* $d \geq 1$ *is a tuple* $\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$*, where:*

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*

– $w_{h_1}, \ldots, w_{h_d}$, *with* $h_1, \ldots, h_d \in \Lambda$, *are strings over* Γ, *describing the initial multisets of objects placed in the* d *regions of* μ;
– R *is a finite set of rules.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

The rules in R are of the following types[1]:

(a) *Object evolution rules*, of the form $[a \to w]_h^\alpha$
They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by w).
(b) *Send-in communication rules*, of the form $a\ [\ ]_h^\alpha \to [b]_h^\beta$
They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h becomes β.
(c) *Send-out communication rules*, of the form $[a]_h^\alpha \to [\ ]_h^\beta\ b$
They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h becomes β.
(d) *Dissolution rules*, of the form $[a]_h^\alpha \to b$
They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.
(e) *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\ [c]_h^\gamma$
They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charges β and γ; the object a is replaced, respectively, by b and c, while the other objects of the multiset originally contained in membrane h are replicated in both membranes.

Notice that "being an elementary membrane" is a *dynamic* property: even if a membrane originally contained other membranes, the dissolution of all of them makes the membrane elementary; if this happens, we also assume that any elementary division rules involving it become applicable, provided that its multiset and charge match the left-hand sides of the rules[2]. Clearly, once a membrane is elementary it can never become non-elementary.

Since it is an essential technical detail in this paper, we carefully distinguish the concepts of *membrane* and *membrane label*, since membrane division allows

[1] The general definition of P systems with active membranes [4] also includes *non-elementary division rules* (type (f)), which are not used in this paper.

[2] The results of this paper, as well as the previous $\mathbf{P}^{\#\mathbf{P}}$ lower bound [8], continue to hold even if we assume that R can only contain elementary division rules for membranes that are *already* elementary in the initial configuration of the P system.

the creation of multiple membranes sharing the same label. In the rest of the paper, we use the expression "membrane h" (singular) only when a single membrane labelled by h is guaranteed to exist, and refer to "membranes (labelled by) h" (plural) otherwise.

Definition 2. *The instantaneous* configuration *of a membrane consists of its label h, its charge α, and the multiset w of objects it contains at a given time. It is denoted by $[w]_h^\alpha$. The* (full) configuration $\mathcal{C}$ *of a P system Π at a given time is a rooted, unordered tree. The root is a node corresponding to the external environment of Π, and has a single subtree corresponding to the current membrane structure of Π. Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h^\alpha$ of the corresponding membranes.*

A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). Analogously, each membrane can only be subject to one communication, dissolution, or elementary division rule (types (b)–(e)) per computation step. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of configurations, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable in $\mathcal{C}_k$.

P systems can be used as language *recognisers* by employing two distinguished objects **yes** and **no**: we assume that all computations are halting, and that either object **yes** or object **no** (but not both) is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. All P systems in this paper are assumed to be confluent.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [2].

Definition 3. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be (polynomial-time) uniform if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:*

- *$F(1^n) = \Pi_n$, where 1^n is the length of x in unary and Π_n is a P system with a distinguished input membrane working on all inputs of length n.*
- *$E(x) = w_x$, where w_x is a multiset encoding the specific input x.*
- *Finally, Π_x is simply Π_n with w_x added to its input membrane.*

The family $\boldsymbol{\Pi}$ is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^\star$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [2] for further details on the encoding of P systems.

Recall that the classes of decision problems solved by uniform and semi-uniform families of P systems working in polynomial time with evolution, communication, dissolution, and elementary division rules are denoted by $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ and $\mathbf{PMC}^\star_{\mathcal{AM}(-\mathrm{n})}$, respectively. When dissolution is not allowed, the two corresponding classes are denoted by $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ and $\mathbf{PMC}^\star_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$.

Finally, we recall the definitions of the complexity classes $\#\mathbf{P}$ and $\mathbf{P}^{\#\mathbf{P}}$ (the latter being equivalent to $\mathbf{P}^{\mathbf{PP}}$), and a notion of completeness for $\#\mathbf{P}$ [3].

Definition 4. *The complexity class $\#\mathbf{P}$ consists of all the functions $f\colon \Sigma^\star \to \mathbb{N}$, also called* counting problems*, with the following property: there exists a polynomial time nondeterministic Turing machine N such that, for each $x \in \Sigma^\star$, the number of accepting computations of N on input x is exactly $f(x)$.*

A function $f \in \#\mathbf{P}$ is said to be $\#\mathbf{P}$-complete under parsimonious reductions *if and only if for each $g \in \#\mathbf{P}$ there exists a polynomial-time reduction $r\colon \Sigma^\star \to \Sigma^\star$ such that $g(x) = f(r(x))$ for each $x \in \Sigma^\star$.*

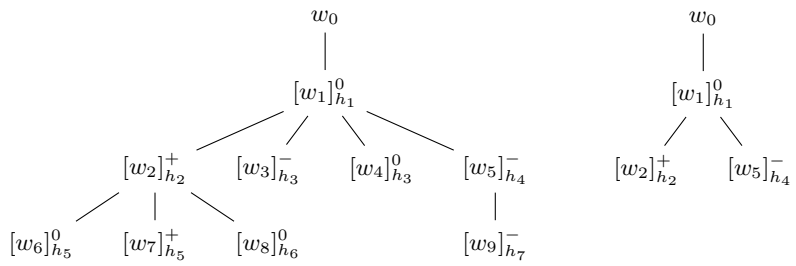


Fig. 1. A full configuration and a partial configuration for the same P system, where in the latter the configurations of all elementary membranes have been removed.

Definition 5. *The complexity class* $\mathbf{P}^{\#\mathbf{P}}$ *consists of all decision problems (languages) recognisable in polynomial time by deterministic Turing machines with oracles for* $\#\mathbf{P}$ *functions. These are Turing machines* M^f*, with* $f \in \#\mathbf{P}$*, having a distinguished* oracle tape *and a* query state *such that, when* M^f *enters the query state, the string* x *on the oracle tape is immediately (i.e., at unitary time cost) replaced with the binary encoding of* $f(x)$*.*

3 The Simulation Algorithm

Let $L \in \mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{n})}$ be a language, and let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a semi-uniform family of confluent recogniser P systems with elementary active membranes deciding L in polynomial time. Let H be a polynomial-time deterministic Turing machine computing the mapping $x \mapsto \Pi_x$.

A straightforward deterministic simulation of $\boldsymbol{\Pi}$, which stores the entire configuration of a P system and applies the rules in a step-by-step fashion, in general incurs an exponential slowdown, due to the exponential size of the configurations of the P systems, even when a binary encoding of the multisets is employed: indeed, by the "Milano Theorem" [10], the exponential space (and time) blowup of the simulation is due to the exponential number of elementary membranes created by division, and not to the number of objects itself. Here we simulate $\boldsymbol{\Pi}$ with a polynomial slowdown by means of a deterministic Turing machine with a $\#\mathbf{P}$ counting oracle.

We begin by observing that we do not need to *explicitly* store the configurations of the elementary membranes (either those that are elementary in the initial configuration, or those that become elementary during the computation), *as long as we can somehow keep track of the interactions between them and their parent regions*. Specifically, we need a way to track the objects that are released from the elementary membranes (via send-out or dissolution rules) and those that are absorbed by them (via send-in rules). We call *partial configuration* the portion of configuration of a P system that we store explicitly, in which we exclude the elementary membranes (unless, temporarily, in the very moment they become elementary and there is only a polynomial number of them) but include

```
1  construct the P system Π_x by simulating H on x
2  let C be the initial (full) configuration of Π_x
3  for each time step t do
4      for each label h in C do
5          if h denotes an elementary membrane then
6              remove its configuration from C
7          else
8              apply the rules to membrane h
9              remove from membrane h the objects sent into
                   elementary children membranes (see Algorithm 2)
10     for each label h′ denoting elementary membranes do
11         add the output of all membranes
               labelled by h′ to their parent multiset
12     if yes or no is found in the environment then
13         accept or reject accordingly
```

Algorithm 1. Simulation of a semi-uniform family $\boldsymbol{\Pi}$ of P systems with elementary active membranes on input x.

the environment surrounding the outermost membrane. An example of partial configuration is shown in Fig. 1.

Since the environment is where the result of any computation of a recogniser P system Π_x is ultimately decided, by the presence of an object **yes** or **no**, an up-to-date partial configuration is sufficient for a Turing machine to establish whether Π_x accepts or rejects. Furthermore, a partial configuration can be stored efficiently when the multisets are encoded in binary, since it consists of a polynomial number of regions containing an exponential number of objects [7].

The main technical result of this paper is a procedure for computing the final partial configuration of a P system in $\boldsymbol{\Pi}$ from the previous partial configurations and the initial full configuration, by using a counting oracle to reconstruct the interactions between the partial configuration and the elementary membranes. This simulation exploits the assumption of confluence of the P systems in $\boldsymbol{\Pi}$ when choosing the multiset of rules to apply at every time step. In principle, the simulation can be adapted to work with any deterministic way of choosing the rules but, in order to simplify the presentation, we give the following priorities:

> *Object evolution rules have the highest priority, followed by send-in rules, followed by the remaining types of rule. Inside these three classes, the priorities are given by any fixed total order.*

Algorithm 1 provides an overview of the simulation; the algorithm takes a string $x \in \Sigma^\star$ as input. First of all, it simulates the machine H that provides the uniformity condition for $\boldsymbol{\Pi}$ on input x, thus obtaining a description of the P system Π_x (line 1). It also stores the initial configuration of $\boldsymbol{\Pi}$ as a suitable labelled tree data structure (line 2).

The main loop (lines 3–13) is repeated for each time step of Π_x, and consists of three main phases. First, the algorithm iterates (lines 4–9) through all membrane labels in the current partial configuration $\mathcal{C}$. The configurations of membranes that have become elementary during the previous computation step are removed from $\mathcal{C}$ (lines 5–6), as described in Section 3.3. Instead, the rules are applied normally [10] to non-elementary membranes (line 8), and the objects that are sent into their children elementary membranes are also removed from them (line 9); the latter operation will be detailed in Section 3.1. In the second phase the algorithm computes the output of the elementary membranes and adds it to their parent membranes (lines 10–11). This phase will be detailed in Section 3.2. Finally, in the third phase (lines 12–13) the algorithm checks whether the object **yes** (resp., **no**) was sent out from the outermost membrane in the current computation step of Π_x; if this is the case, the simulation halts by accepting (resp., rejecting) x.

3.1 Moving Objects *into* Elementary Membranes

Line 9 of the simulation algorithm requires the removal from a specific non-elementary membrane h of the objects sent into its elementary children membranes. While the current configuration of h is known, as it is stored in the partial configuration $\mathcal{C}$, the configuration of its children membranes is not stored.

We solve this problem by nondeterministically simulating each elementary children membrane of h having label h'; each computation keeps track of just a single membrane h' among those obtained by division, starting from the moment the initial (unique) membrane h' became elementary, up to the current time step. In order to do so, whenever a membrane becomes elementary (a fact discovered at line 5 of the simulation algorithm), before removing it from $\mathcal{C}$ the algorithm stores its label, its configuration, and the current time step t; this only requires polynomial space, since at most one configuration per label is stored.

We are only interested in simulating the *internal* configuration of elementary membranes (i.e., the multiset it contains and its charge), without keeping track of any objects released from them either by send-out or dissolution rules. Hence, the only problematic rules are those of type send-in, since multiple membranes sharing the same parent compete for the same objects, but cannot coordinate because each membrane is simulated by a different computation; recall that distinct computations of a nondeterministic Turing machine do not communicate. In order to solve the conflicts, a table $\mathsf{apply}[r, t]$ can be precomputed, associating each send-in rule $r = a\ [\ ]_h^\alpha \to [b]_h^\beta$ and time step t with the set of individual membranes with label h that must apply r at time t.

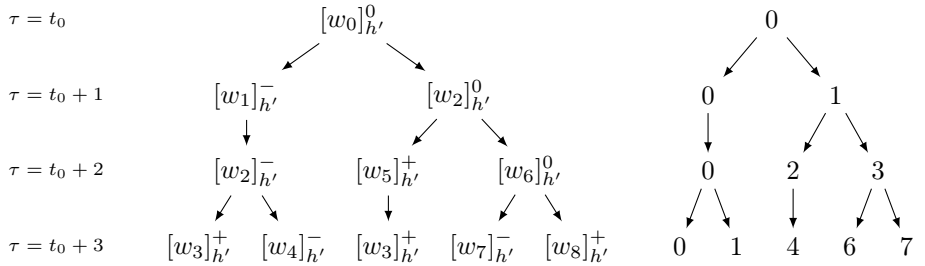
Computing the entries of the table apply first requires us to name each individual membrane among those sharing the same label. In order to do so, we exploit a solution already proposed by Sosík and Rodríguez-Patón [9], attaching to each elementary membrane an identifier computed as follows:

- When membrane h becomes elementary, its identifier is $id = 0$.
- If a membrane has identifier id at time τ and no division occurs at that time, its new identifier at time $\tau + 1$ is $2 \times id$.

– If a membrane has identifier id at time τ and a division occurs at that time, the new identifiers of the two resulting membranes at time $\tau + 1$ are $2 \times id$ and $2 \times id + 1$, respectively.

This ensures that, at time t, each membrane with label h has a unique identifier in the range $[0, 2^t - 1]$ (although not all identifiers in that range need to correspond to actual membranes).

Example 1. The procedure that assigns unique identifiers to elementary membranes sharing the same label can be represented graphically. On the left, we show a tree of membrane divisions (not to be confused with a configuration tree, as illustrated in Fig. 1): time goes downward, starting from the moment t_0 when the membrane became elementary, each level representing a time step; notice that not all membranes necessarily divide at each step. On the right, we show the identifiers assigned to the membranes:



Notice how all identifiers differ, even for membranes having the same configuration: for example, the two copies of $[w_3]^+_{h'}$ at time $\tau = t_0 + 3$ have identifiers 0 and 4. Furthermore, not all identifiers between 0 and $2^t - 1$ correspond to a membrane: here 2, 3, and 5 do not correspond to any membrane.

Naively storing arbitrary subsets of $[0, 2^t - 1]$ as entries of the table apply requires an exponential amount of space with respect to t. We can, however, exploit the hypothesis of confluence of the family $\boldsymbol{\Pi}$ and choose, among all possible computations of the simulated P system, the computation where each send-in rule $r = a\ [\ ]^\alpha_h \to [b]^\beta_h$ is applied, at each step t, to all (and only the) membranes h having charge α whose identifiers are included in an interval, to be computed as described below. Distinct intervals involving the same membrane label h may overlap only for rules r_1, r_2 having a different charge on the left-hand side, as shown in the following example.

Example 2. Consider the P system of Example 1 at time $t = 3$, and suppose the following rules (sorted by priority) are associated to the membrane label h':

$$r_1 = a\ [\ ]^-_{h'} \to [c]^+_{h'} \qquad r_2 = a\ [\ ]^+_{h'} \to [d]^+_{h'} \qquad r_3 = b\ [\ ]^+_{h'} \to [e]^0_{h'}$$

Furthermore, suppose the parent membrane h of h' contains exactly 3 instances of object a and 4 instances of b. Then, one of the possible choices of intervals of

e1 set $id \coloneqq 0$
e2 **for** each time step $\tau \in \{t_0, \ldots, t-1\}$ **do**
e3 set $newid \coloneqq 2 \times id$
e4 **for** each applicable evolution rule $r \in R$ involving h' **do**
e5 apply r as many times as possible
e6 **for** each non-evolution rule r applicable to h' **do**
e7 **if** $r = [a]^{\alpha}_{h'} \to [\,]^{\beta}_{h'}\ b$ **then**
e8 remove an instance of a from the membrane
e9 change the charge of the membrane to β
e10 **else if** $r = [a]^{\alpha}_{h'} \to b$ **then**
e11 **reject**
e12 **else if** $r = [a]^{\alpha}_{h'} \to [b_0]^{\beta_0}_{h'}\ [b_1]^{\beta_1}_{h'}$ **then**
e13 nondeterministically guess a bit i
e14 set $newid \coloneqq newid + i$
e15 rewrite an instance of a to b_i
e16 change the charge of the membrane to β_i
e17 **else if** $r = a\ [\,]^{\alpha}_{h'} \to [b]^{\beta}_{h'}$ and $\mathsf{left}[r,\tau] \leq id \leq \mathsf{right}[r,\tau]$ **then**
e18 add an instance of b to the membrane
e19 change the charge of the membrane to β
e20 set $id \coloneqq newid$

Algorithm 2. Nondeterministic simulation of steps t_0 to $t-1$ of an elementary membrane h'. Here t_0 is the time when membrane h' became elementary.

identifiers, which is consistent with the previous discussion, is

$$\mathsf{apply}[r_1,t] = [0,7] \qquad \mathsf{apply}[r_2,t] = [0,3] \qquad \mathsf{apply}[r_3,t] = [4,7]$$

The interval $\mathsf{apply}[r_1,t]$ contains exactly 2 identifiers of membranes h' having negative charge (the maximum number of instances of a that can be sent in by means of rule r_1); the interval $\mathsf{apply}[r_2,t]$ contains only 1 identifier of membranes h' with positive charge (since only one copy of a remains that is not used up by rule r_1); finally, the interval $\mathsf{apply}[r_3,t]$ contains the remaining 2 identifiers of membranes h' with positive charge, leaving two instances of b unused by send-in rules. Notice that the interval $\mathsf{apply}[r_1,t]$ overlaps with the others, since rule r_1 is applied to membranes h' having different charge than those where r_2 and r_3 may be applied, while $\mathsf{apply}[r_2,t]$ and $\mathsf{apply}[r_3,t]$ are necessarily disjoint.

We can store each interval $\mathsf{apply}[r,t]$ as the pair of bounds $\mathsf{left}[r,t]$ and $\mathsf{right}[r,t]$. This only requires two binary numbers per each original entry $\mathsf{apply}[r,t]$; hence, the information contained in the table can now be stored in polynomial space.

The pseudocode for the nondeterministic simulation of the elementary membranes labelled by h', starting from the time t_0 when h' became elementary and up to time $t-1$ (i.e., leaving out the last computation step), is shown as Algorithm 2. This is a straightforward simulation algorithm for P systems, as already described in the literature [10, 7], except for a few key differences:

– the simulation aborts by rejecting if the membrane dissolves within step $t-1$ (lines e10–e11);
– when simulating an elementary division (lines e12–e16) the algorithm performs a nondeterministic choice between the two membranes resulting from the division, and continues to simulate only one of them;
– before simulating a send-in rule (lines e17–e19), the algorithm checks whether the identifier of the membrane being simulated belongs to the range of identifiers of membranes applying that rule at time τ.

Notice that the loop of lines e6–e19 applies at most one non-evolution rule, as required by the semantics of P systems with active membranes.

Let $r = a\,[\,]^{\alpha}_{h'} \to [b]^{\beta}_{h'}$ be a send-in rule. We can simulate a further computation step of each membrane labelled by h', halting by accepting if rule r is actually applicable (i.e., the current membrane h' has charge α, and its identifier is in the correct range), and rejecting otherwise. As a consequence, Algorithm 2 with this additional step has as many accepting computations as the number of membranes having label h' where rule r is applied at time t. This proves the following:

Lemma 1. *Suppose we are given the configuration $[w]^{\alpha}_{h'}$ of an elementary membrane h', a set of rules, two time steps t_0 (corresponding to configuration $[w]^{\alpha}_{h'}$) and t expressed in unary notation*[3]*, two tables* left *and* right *as described above, and a specific send-in communication rule $r = a\,[\,]^{\alpha}_{h'} \to [b]^{\beta}_{h'} \in R$. Then, counting the number of applications of rule r at time t is in* **#P**. □

Remark 1. Counting the number of satisfying assignments for a Boolean formula in 3CNF (a **#P**-complete problem) can be reduced in polynomial time to counting the number of send-in rules applied during the "counting" step by the P systems solving the Threshold-3SAT problem described in [6]. This requires setting the threshold to 2^m, where m is the number of variables. Hence, the counting problem of Lemma 1 is actually **#P**-complete.

Line 9 of Algorithm 1 can thus be expanded as Algorithm 3, where the entries of left and right are also computed. The number of objects sent in from h to its children membranes h' by means of a rule $r = a\,[\,]^{\alpha}_{h'} \to [b]^{\beta}_{h'}$ can be computed (line 9.8) as the minimum k between the number of instances of a contained in h and the number m of membranes h' having charge α which have not yet been assigned a send-in rule. The initial value of m is thus the number of membranes h' having identifier at least $\mathsf{left}[r,t]$ and charge α. The membranes h' having charge α and an identifier smaller than $\mathsf{left}[r,t]$ have already been assigned to a different interval, and will apply a different send-in rule. The algorithm can then remove k copies of a from h (line 9.9). The value of $\mathsf{right}[r,t]$ must then be updated, in order to ensure that exactly k membranes with label h' apply r at time t. A correct value for $\mathsf{right}[r,t]$ can be determined by performing a binary search in the interval $\left[\mathsf{left}[r,t], 2^t - 1\right]$ while recomputing the value of m (line 9.12) as in Lemma 1.

[3] Expressing the number of time steps t in unary is necessary for the problem to be in **#P**, otherwise the value t might be exponentially larger than its representation, thus increasing the complexity of the problem.

9.1 **for** each elementary membrane label h' contained in h **do**
9.2 **for** each charge $\alpha \in \{+, 0, -\}$ **do**
9.3 set $\ell := 0$
9.4 **for** each rule $r = a\,[\,]^{\alpha}_{h'} \to [b]^{\beta}_{h'}$ in R **do**
9.5 set $\mathsf{left}[r, t] := \ell$
9.6 set $\mathsf{right}[r, t] := 2^t - 1$
9.7 let m be the number of membranes with label h' and $\mathsf{left}[r, t] \leq id \leq \mathsf{right}[r, t]$ where r is applicable at time t
9.8 set $k := \min\{m, \text{number of instances of } a \text{ in } h\}$
9.9 remove k instances of a from h
9.10 **while** $m \neq k$ **do**
9.11 update $\mathsf{right}[r, t]$ by binary search
9.12 recompute m
9.13 set $\ell := \mathsf{right}[r, t] + 1$

Algorithm 3. Removing from a non-elementary membrane h the objects sent into its elementary children membranes (Algorithm 1, line 9).

3.2 Moving Objects *from* Elementary Membranes

Line 11 of Algorithm 1 deals with communication in the opposite direction with respect to line 9, i.e., from the elementary membranes towards their parent.

For each label h' denoting an elementary membrane and for each object type a, the number of instances of a released from membranes with label h' at time t can be determined by first simulating these membranes up to time $t - 1$ by using Algorithm 2; the last time step is then simulated as follows:

- If a rule sending out a is applied, the computation accepts.
- If the membrane dissolves, the algorithm accepts as many times as the number k of instances of a in the simulated membrane. This requires "forking" k accepting computations, which in nondeterministic Turing machines corresponds to first nondeterministically guessing a number between 1 and k, and then accepting. This can be performed in polynomial time even if k is exponential, since the number of bits of k is polynomial.
- Otherwise, the computation rejects.

The number of accepting computations of the algorithm is the number of instances of a to be added to the parent of h'. This proves the following:

Lemma 2. *Suppose we are given the configuration $[w]^{\alpha}_{h'}$ of an elementary membrane h', a set of rules, two time steps t_0 (corresponding to configuration $[w]^{\alpha}_{h'}$) and t expressed in unary notation, two tables* left *and* right *as described above, and an object type a. Then, counting the number of instances of object a released (via send-out or dissolution rules) from membranes with label h' at time t is in* **#P**. □

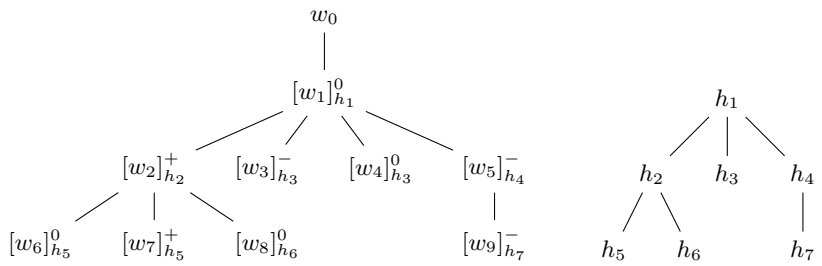


Fig. 2. A full configuration and the tree $\mathcal{T}$ containing the information about the inclusions among its membranes, expressed in terms of labels only.

Remark 2. We can reduce the assignment counting problem of Remark 1 to counting the number of objects sent out by elementary membranes. Indeed, during the "checking" phase of the aforementioned algorithm for Threshold-3SAT [6], all membranes containing a satisfying assignment simultaneously send out a specific object. Hence, the counting problem of Lemma 2 is also **#P**-complete.

3.3 Deciding if a Membrane is Elementary

In order to decide (line 5 of Algorithm 1) if a label h belongs to an elementary membrane at time t (recall that membranes sharing the same label are either all elementary and share the same parent, or there is only one of them) without keeping them in the partial configuration $\mathcal{C}$, we can use an auxiliary data structure. We employ a rooted, unordered tree $\mathcal{T}$ whose nodes are labels from Λ, i.e., no repeated labels appear in $\mathcal{T}$. This tree represents the inclusion relation between membranes before each simulated step, and it is initialised before entering the main loop of the simulation (lines 3–13). As an example, Fig. 2 shows a full configuration and the corresponding inclusion tree between membranes labels. This data structure is updated in two different steps of Algorithm 1: at line 8, when non-elementary membranes dissolve, and after the execution of line 5 (as described below).

Then, for each label h appearing in the partial configuration $\mathcal{C}$ (line 4), either it is associated to an internal node in $\mathcal{C}$, and in that case it is not elementary, or it is a leaf in $\mathcal{C}$, and then the node h has leaf children in $\mathcal{T}$. In that case, since we do not keep track of the elementary membranes in partial configurations, the algorithm needs to explicitly check, for each child membrane label h' of h, whether there does actually still exist at least one membrane with label h'.

Once again, we employ Algorithm 2 to simulate the elementary membranes with label h' up to time step $t-1$; if the algorithm has not already rejected by then, this means that the simulated membrane still exists, and the computation must accept. In this case there is no need to count the number of accepting computations, but only to check whether there exists at least one. We can thus state the following:

Lemma 3. *Suppose we are given the configuration $[w]^{\alpha}_{h'}$ of an elementary membrane h', a set of rules, two time steps t_0 (corresponding to configuration $[w]^{\alpha}_{h'}$) and t expressed in unary notation, and two tables* left *and* right *as described above. Then, deciding whether there exists a membrane with label h' at time t is in* **NP**. □

Remark 3. The most common solutions to **NP**-complete problems using P systems with elementary active membranes (such as [10]) first generate one membrane per candidate solution, then check all of them for validity. By dissolving all membranes containing invalid solutions, we can easily show that the decision problem of Lemma 3 is actually **NP**-complete by reduction from any other **NP**-complete problem.

4 A Characterisation of $\mathbf{P}^{\#\mathbf{P}}$

We are now able to analyse the complexity of the simulation algorithm.

Lemma 4. *Algorithm 1 runs in polynomial time on a deterministic Turing machine with access to an oracle for a* **#P***-complete function.*

Proof. Lines 1 and 2 can be executed in polynomial time due to the assumption of semi-uniformity of the family $\boldsymbol{\Pi}$. The loop of line 3 simulates a polynomial number of time steps, and those of lines 4 and 10 iterate over sets of membrane labels. Hence, lines 5–9 and 11–13 are executed a polynomial number of times.

Line 6 simply consists in removing a node from a tree data structure, line 8 in the application of rules to a polynomial number of membranes, which can be performed in polynomial time [10, 7], and lines 12–13 in a membership test.

Let $f\colon \Sigma^\star \to \mathbb{N}$ be **#P**-complete under parsimonious reductions (e.g., #SAT). Then, the counting problems of Lemmata 1 and 2 and the decision problem of Lemma 3 can be reduced in polynomial time to evaluating f. Suppose we have access to an oracle for f.

The implementation of line 5 described in Section 3.3 employs the auxiliary tree data structure $\mathcal{T}$ (which can be scanned and updated in polynomial time), and can ascertain the existence of children membranes of h by performing a reduction and a query to the oracle for f at most once per label.

Moving objects into elementary membranes (line 9), as described in Section 3.1, executes Algorithm 3. This performs a polynomial number of iterations of lines 9.5–9.13. Lines 9.7 and 9.12 are reductions followed by an oracle query for f. The loop of lines 9.10–9.12 is executed a logarithmic number of times with respect to the exponential number of identifiers (i.e., a polynomial number).

Finally, line 11 of Algorithm 1, whose implementation is described in Section 3.2, computes the output of the elementary membranes. This also consists in performing a reduction and an oracle query to f for each elementary membrane label and each object type. The statement of the lemma follows. □

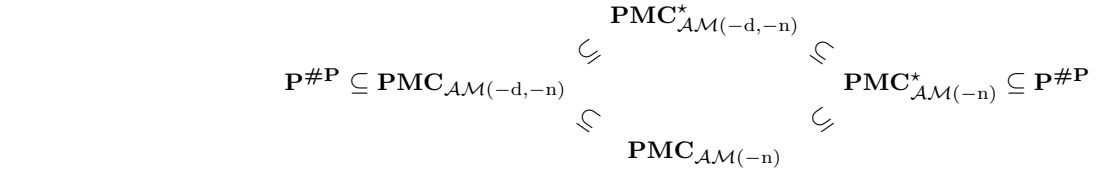
As a consequence, we obtain the equivalence of several complexity classes for P systems with elementary active membranes to $\mathbf{P}^{\#\mathbf{P}}$.

Theorem 1. *The following equalities hold:*

$$\mathbf{PMC}^{[\star]}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} = \mathbf{PMC}^{[\star]}_{\mathcal{AM}(-\mathrm{n})} = \mathbf{P}^{\#\mathbf{P}}$$

where $[\star]$ *denotes optional semi-uniformity instead of uniformity.*

Proof. The inclusions of the following diagram hold:



Indeed, uniformity implies semi-uniformity, and adding dissolution rules does not decrease the power of P systems. Furthermore, P systems with restricted (without dissolution) elementary active membranes are able to efficiently simulate polynomial-time deterministic Turing machines with counting oracles [8]. The last inclusion $\mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{n})} \subseteq \mathbf{P}^{\#\mathbf{P}}$ follows from Lemma 4. □

4.1 Consequences for the P Conjecture

In P systems without charges, the presence of dissolution rules possibly makes a difference. Without them, only problems in **P** may be solved in polynomial time [1], and the P conjecture [5, Problem F] claims that not even elementary division together with dissolution rules can break this barrier. However, the tightest known upper bound up to now was **PSPACE** [9].

P systems without charges are equivalent to P systems *with* charges where the membranes always remain neutral (i.e., no rule ever changes a membrane charge). Hence, the class of problems solved in polynomial time by the former model without non-elementary division rules, denoted by $\mathbf{PMC}_{\mathcal{AM}^0(-\mathrm{n})}$, is trivially included in $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$. The corresponding inclusion also holds in the semi-uniform case. As a consequence, Theorem 1 implies the following improved upper bound for the P conjecture:

Corollary 1. $\mathbf{PMC}^{[\star]}_{\mathcal{AM}^0(-\mathrm{n})} \subseteq \mathbf{P}^{\#\mathbf{P}}$. □

5 Conclusions

In this paper we have presented a simulation of polynomial-time semi-uniform families of P systems with elementary active membranes, characterising the complexity class $\mathbf{PMC}^{\star}_{\mathcal{AM}(-\mathrm{n})}$, by means of deterministic Turing machines working in polynomial time with access to a **#P** oracle. This simulation and the previously known lower bound [8] complete the characterisation of this complexity class, as well as those obtained by requiring uniformity instead of semi-uniformity or disallowing dissolution rules: all these classes coincide with $\mathbf{P}^{\#\mathbf{P}}$. This result is

also interesting because it represents an improvement of the upper bound of the computational power for the class of P systems involved in the P conjecture. We hope that this step will help in the search for a solution to the conjecture.

In the future we plan to investigate the computational power of P systems with active membranes in which non-elementary division is allowed but limited to membranes of a certain depth. While unrestricted non-elementary division increases the computational power to **PSPACE**, we conjecture that limited division can generate a hierarchy of complexity classes. Furthermore, we plan to investigate the computational power of non-confluent P systems with active membranes using division rules. Currently, no upper bound tighter than **NEXP** is known for these classes of P systems.

References

1. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Nuñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. International Journal of Computer Mathematics 83(7), 593–611 (2006)
2. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
3. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
4. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
5. Păun, Gh.: Further twenty six open problems in membrane computing. In: Gutíerrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 249–262. Fénix Editora (2005)
6. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Elementary active membranes have the power of counting. International Journal of Natural Computing Research 2(3), 329–342 (2011)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. International Journal of Foundations of Computer Science 22(1), 65–73 (2011)
8. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems simulating oracle computations. In: Gheorghe, M., Păun, Gh., Salomaa, A., Rozenberg, G., Verlan, S. (eds.) Membrane Computing, 12th International Conference, CMC 2011, Lecture Notes in Computer Science, vol. 7184, pp. 346–358. Springer (2012)
9. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
10. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)